SAFETY IN NUMBERS

Redundancy lets a system perform its intended functions despite some number of faults

VICTOR P. NELSON

Fault-tolerant computing can increase the dependability of a computer system by providing more hardware, software, or information than is necessary. This redundancy lets a system perform its intended functions despite some number of faults.

Quantitatively, you can measure how dependable a system is in terms of either reliability or availability. System *reliability* is the probability that the system won't fail by a given time. If a system needs continuous error-free operation, a minimum reliability level must be maintained over the system's useful lifetime.

A system's maximum useful lifetime is the length of time in which its reliability remains greater than some specified minimum value. Fault-tolerant computing is one method for increasing a system's useful lifetime.

It's important to note two things, however. First, for a given application, a system could be sufficiently reliable without fault tolerance. And second, using fault tolerance doesn't necessarily guarantee that a system will be sufficiently reliable for a particular application.

If occasional, brief periods of downtime are acceptable in an application, an availability goal may be more appropriate. *Availability* is the probability that a system will be operational at any given moment; thus, it is the ratio of the system's uptime to the sum of its uptime and downtime. Availability is increased by using fault-tolerant designs to maximize uptime or minimize downtime.

ILLUSTRATION: JOHN LABBE © 1991

Faults and Errors

The words *fault* and *error* sound like synonyms; they're not, however, and the

distinction between them is important. A fault is a physical condition that occurs in a hardware or software element, making the element unable to perform its intended function. An error, on the other hand, is a symptom of a fault and manifests itself as an incorrect output or invalid state for the faulty element.

A fault is referred to as *latent* when it occurs without producing errors during system operation. A common example would be a fault that alters the contents of a byte in memory. If the byte is not accessed after this change, no error occurs.

You can characterize faults by duration and extent. Fault *duration* may be permanent, transient, or intermittent. A *permanent* fault doesn't disappear once it occurs. It results from failures of electronic components or interconnections, physical damage, or design errors. Design errors are especially difficult to detect, since the affected hardware or software often performs as designed.

Transient faults are temporary conditions, usually the result of electromagnetic interference, temperature, humidity, incorrect operating voltage, or other external disturbances. Transient faults typically disappear as soon as the external condition is eliminated.

An *intermittent* fault alternates between active and dormant states and is usually caused by poor design, borderline operating conditions, or the marginal operation of a component prior to failure. For both transient and intermittent faults, errors may remain after the fault disappears.

Many systems use diagnostic programs to locate faults. However, these diagnostics are only effective for permanent faults. Systems need to use other methods to locate transient and intermittent faults, which are far more likely to occur in real systems.

The *extent* of a fault indicates how much of the system it affects. A local fault directly affects a single component, while a global fault influences multiple components. Most fault-tolerance strategies deal with a limited number of localized faults; they tend to leave systems vulnerable to global faults.

Transient faults associated with external disturbances tend to be global in nature, since the entire system is typically exposed to the same condition. In contrast, the failure of a transistor junction would directly affect only the component that contains it.

Campaign Strategies

The ability to tolerate faults requires a design strategy that includes one or more of the following elements: error detection, error masking, error confinement, fault diagnosis, system repair and reconfiguration, and system recovery.

To detect errors, you need to have enough redundancy that the system can distinguish between correct and incorrect information. You can create redundant information by replicating the modules that produce the information, by encoding the information so that errors result in detectable noncode words, or by using heuristics to determine whether the information is valid or reasonable (e.g., a square-root algorithm that produces a negative result would be faulty).

For continuous error-free operation, the system must dynamically correct or mask errors. Error masking requires more redundancy than error detection does, because the system must extract the correct information from the information that the redundant configuration produced. Error masking also typically uses duplicate modules or extra bits to encode information.

You can often mask the effects of transient faults simply by retrying the operation that failed. The bus interfaces of several current microprocessors let externally detected errors initiate bus-cycle retries transparent to the software.

To minimize the impact of a fault, you must establish error-containment boundaries to confine errors to their originating modules. You don't want them to propagate through the rest of the system. Error-containment boundaries prevent errors from spreading into or out of a module by checking all its inputs and outputs, respectively, and then isolating

BYTE ACTION SUMMARY

With fault-tolerant computing, you can increase your system's dependability by selectively providing more hardware, software, or information than you need. You can also increase your system's useful lifetime. Fault tolerance means being able to detect, mask, and confine errors; diagnose faults; and repair, reconfigure, and recover your system. What is the key? Redundancy.

the module from the rest of the system if an error is found.

In systems that have enough resources to continue operating without one or more modules, you must not let a failed module affect the remaining resources. Whether or not the system can continue, limiting error propagation minimizes the amount of time needed to repair any damage.

To repair the system, you must first analyze the errors to identify which components are faulty. How detailed a diagnosis you need depends on your system repair and reconfiguration strategy. If you plan to replace faulty modules, whether automatically or manually, you just identify which module is faulty.

You don't gain any benefit by analyzing faults further unless you plan to repair faulty modules; for example, the diagnostic programs of the Bell System's 1A Processor, which was the heart of the company's first electronic-switching systems, focused on isolating a problem only to the three replaceable modules it might occur in. This broad-brush approach minimized repair time and increased system availability.

In a fault-tolerant system, you must either replace a faulty component or route information around it to keep it from interfering with how the rest of the system operates. In most commercial and industrial applications, repair is manual; circuit boards are replaced by hand.

Some commercially available fault-tolerant systems incorporate a "hot repair" capability. This lets you deenergize a faulty board and remove it from the system, install and energize a spare board, and integrate the new board into the system, all without bringing the system down. The rest of the system can continue operations during this process.

Where it's not practical to repair a system manually—such as in space vehicles or aircraft during flight operations—reconfiguration must occur automatically. The system must be able to isolate the faulty module by switching off its power or otherwise segregating its outputs from the rest of the system. Once the system isolates the faulty module, it can switch on a spare module to replace the faulty one, or it can transfer the module's tasks to another operational unit.

If errors have propagated in a system, if new hardware is introduced, or if work has been transferred between modules, you may have to restore the system's state or set it to some acceptable value before operations can continue.

System recovery can be either forward or backward. To implement *backward recovery*, the system saves its state at various checkpoints. After repair or reconfiguration, the system's state is restored to that of the last good checkpoint, and all processing is repeated from that point. It's important in backward recovery to identify those operations that can't be repeated, such as posting a deposit to a customer's bank account.

When errors have not been significantly propagated through the system, you can implement *forward recovery* by masking errors or otherwise deriving a correct system state following the occurrence of a fault. System operation can then simply continue without having to roll back to an earlier state. You would initialize any new hardware introduced during repair or reconfiguration to the current state of the system prior to continuing.

But if error propagation has been more significant, further recovery actions may be necessary. You may have to undo an interrupted database update to put the database in a consistent state. Or you may have to reacquire an object that a radar system was tracking. To minimize recovery time, it is critical that you enforce error-containment boundaries.

Multiple Modules

Active/backup module pairs. The most common form of modular redundancy is to simply replace a faulty module with a spare (see figure 1a). To do this, the active module must include internal error-detection mechanisms, and the system must properly transfer program control to the backup module.

The backup unit can be hot or cold. A *hot* spare performs all computations in parallel with the active unit, and thus it always contains the correct state of the system, making the switchover instantaneous. A *cold* spare can be either unpowered or used for other work until the active module fails.

Tandem Computer's (Cupertino, CA) NonStop systems use active/backup process pairs. The backup process remains dormant, letting the computer perform other tasks. The active module sends state information to the backup at various process checkpoints, so the backup can initiate execution from the last checkpoint if the active module fails. A cold spare, which is used for other work, increases overall production, but at the expense of a longer transition time for the backup to replace the failed unit.

Duplex operations. You can often detect errors more completely by using identical modules in a duplex configuration (see figure 1b). In this approach, two modules perform all operations in lock-step fashion, and they use a *comparator* to detect any mismatch between

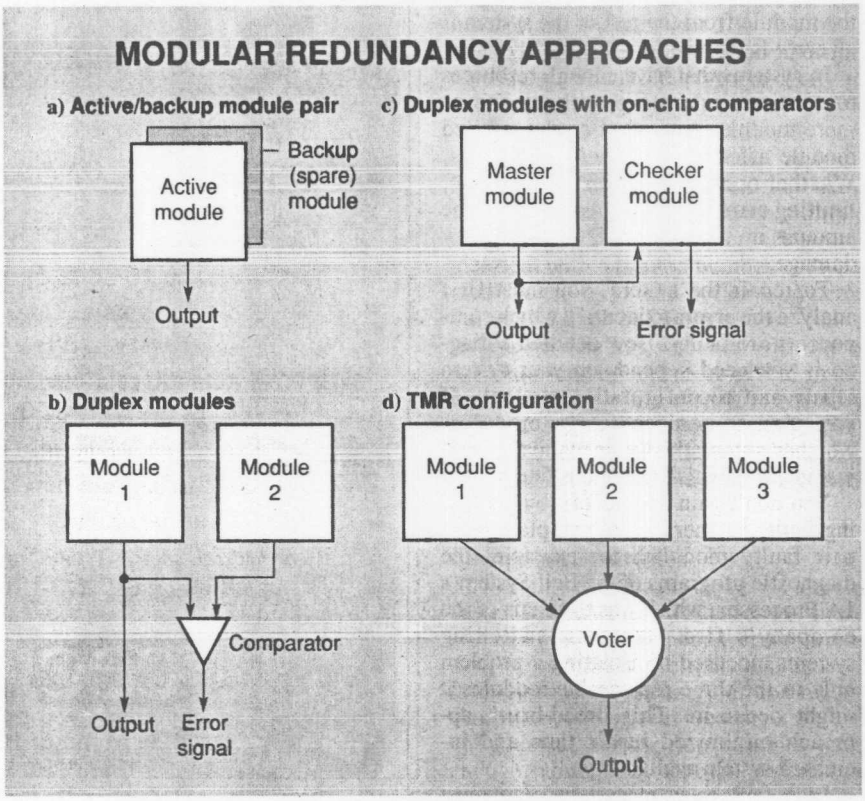

Figure 1: (a) *The active/backup module pair simply replaces a faulty module with a spare.* **(b)** *Duplex operations use two identical modules performing all operations and a comparator to detect any mismatch in outputs.* **(c)** *Duplex operations with on-chip comparators provide comparators at each output pin and master/checker operating mode.* **(d)** *In the triple-modular-redundancy (TMR) configuration, three identical modules perform each operation concurrently, and a voter determines a majority ruling, thus masking failures in any one module.*

the two sets of outputs.

As soon as the system detects an error, it disables the outputs of the module and issues an error signal. You can then discard the entire duplex module and reassign its tasks, or you can run additional diagnostics to determine which of the two units in the module is faulty so that the good unit can continue on its own.

Bell's 1A Processor used two identical processors that brought 12 internal points out to comparators—two points during each clock cycle. If an error was detected, the diagnostics selected one of the two processors to continue operations until repairs could be made.

Duplex operations with on-chip comparators. Several recent VLSI devices, including the Intel APX-432 microprocessor family and the AMD 29000 RISC processor family, have incorporated support for on-chip duplex operations. Each device includes comparators at each output pin and master/checker operating mode.

One chip in each pair is designated the master and drives all outputs normally. The second chip, designated the checker, disables its output drivers and samples the outputs that the master chip supplies. The on-chip comparators within the checker detect any disagreements between the two chips and provide the error signal (see figure 1c).

Process outputs can also be compared in software, allowing the two standard modules to operate in a loosely coupled duplex configuration. Typically, the two processes would exchange all critical information and compare the two copies in software prior to using that information.

Triple-modular redundancy (TMR). Duplex configurations detect errors without identifying which module is correct or faulty. If you need continuous real-time operations, you do not have the time to stop the system to find out which unit is correct. Continuous operation requires that the system mask errors instantaneously. Repair and reconfiguration operations have to take place either in parallel with normal operations or later

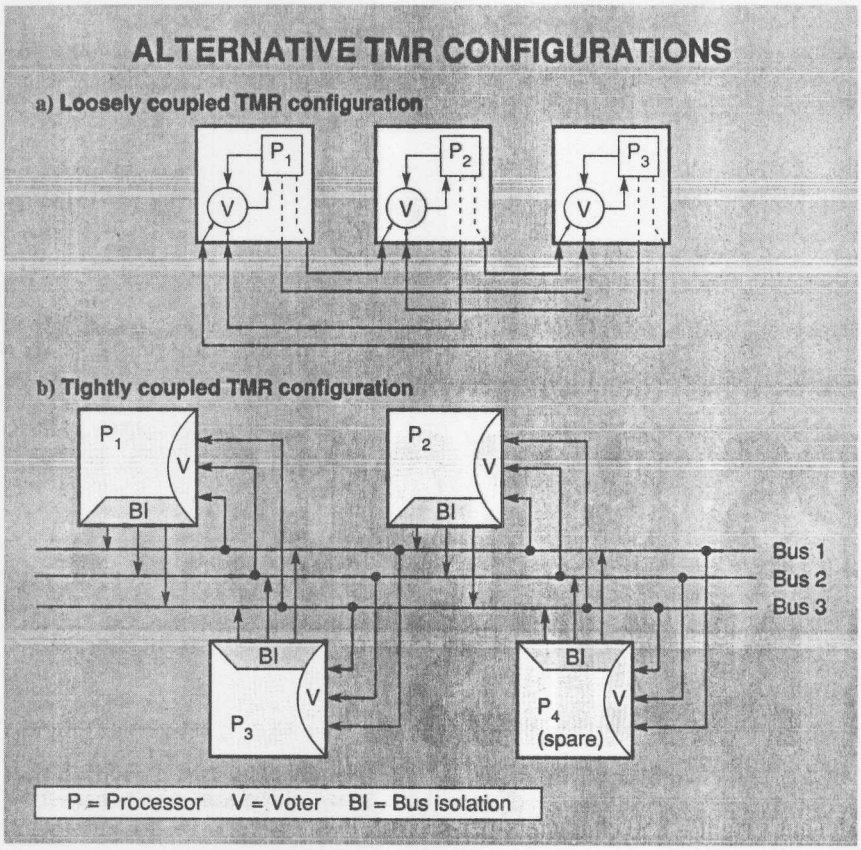

Figure 2: (a) *In a loosely coupled TMR configuration, software performs the voting.* **(b)** *In a tightly coupled TMR configuration, processors are each assigned to drive one bus through their bus-isolation logic and read all three buses via their input voters.*

at a more convenient time.

TMR is the most common fault-masking configuration in which three identical modules perform each operation concurrently (see figure 1d). A voter selects the overall output to correspond to the majority vote of the three modules, thus masking failures in any one of them. You can readily extend this process to *n*-modular redundancy with *n* identical modules and a corresponding majority voter.

As in duplex operations, the vote in an *n*-modular redundant configuration can be performed in hardware, on a cycle-by-cycle basis, or in software, with modules exchanging data and using software voting (see figure 2a). TMR is used in the Space Shuttle computer complex (which uses four processors) and in the experimental SIFT aerospace computer and its commercial counterpart, the August Systems industrial process-control computer (three processors). These systems exchange all critical data values and vote on them before they are used in any program step.

While the TMR configuration in figure 1d tolerates any processor failure, it is still vulnerable to voter failure. If the probability of voter failure is significant, you can use three voters; for example, figure 2b shows the configuration of the Fault-Tolerant Multiprocessor (FTMP) developed at Charles Stark Draper Labs (Cambridge, MA) for aerospace applications.

In FTMP, you can group any three processors into a TMR triad, with each processor driving one of the redundant buses, reading all three buses, and voting on their inputs; thus, the failure of any processor or any voter disables the entire processor/voter pair. You can assign any other processor to replace the failed unit within the affected triad simply by telling it which bus to drive.

Self-checking module pairs. One alternative to voting that can perform continuous error-free operation is to use self-checking module pairs. The quadruplex configuration (see figure 3) has been used in the 68000-based Stratus 32 systems (Stratus Computer, Marlborough, MA).

In the quadruplex configuration, two pairs of duplex modules (a total of four processors and two comparators) perform all operations concurrently. Each duplex module is self-checking in that a comparator detects any disagreements between its two processors. If such an error occurs, the system disables that module's output, and while it replaces the faulty module, the remaining duplex module continues operating alone.

Several techniques are used to design modules that are self-checking but not replicated. The Bell System's 3A Processor (successor to the 1A) uses two processors that operate autonomously except for periodically exchanging state information. Coding and self-checking logic within each processor enable a faulty processor to identify itself; when that happens, the second processor takes over, providing continuous error-free system operation.

Information redundancy. Applying coding techniques to redundant bits can make it easier to detect and correct errors within an information word. Error-detecting and -correcting codes are the most widely used form of fault tolerance, with applications ranging from aerospace and military systems to laptop personal computers.

The main attraction of coding is that it can detect and correct errors with significantly less redundancy than you find with replicated modules. However, most coding schemes apply only where information is not transformed, such as in information storage or retrieval (e.g., memory, disk, and tape) and in data transmission over buses or communications channels.

How well a coding scheme detects or corrects errors depends on how well it can sort out the valid code words. A given number of errors must not be able to transform one valid code word to another; it must turn the code word into a noncode word. With additional redundancy, the separation between the two can be wide enough to associate specific noncode words with specific code words. When this occurs, a limited number of errors can be corrected.

The separation between two binary words—referred to as the *Hamming* distance—is defined as the number of bit positions in which the two words differ. Suppose two valid code words differ in a single bit position (i.e., they have a separation of one). An error in that single bit position will transform one valid code word into another, and the error will be undetectable.

If the minimum separation is two, then

for any valid code word, a single error can only produce a noncode word, and the error can be found. However, if two errors were to occur, one valid word could be converted to another valid word, and the errors would not be seen.

If the minimum separation increases to three, however, each single error produces a noncode word that can be uniquely associated with its original code word. When this occurs, the system can produce the correct data during decoding.

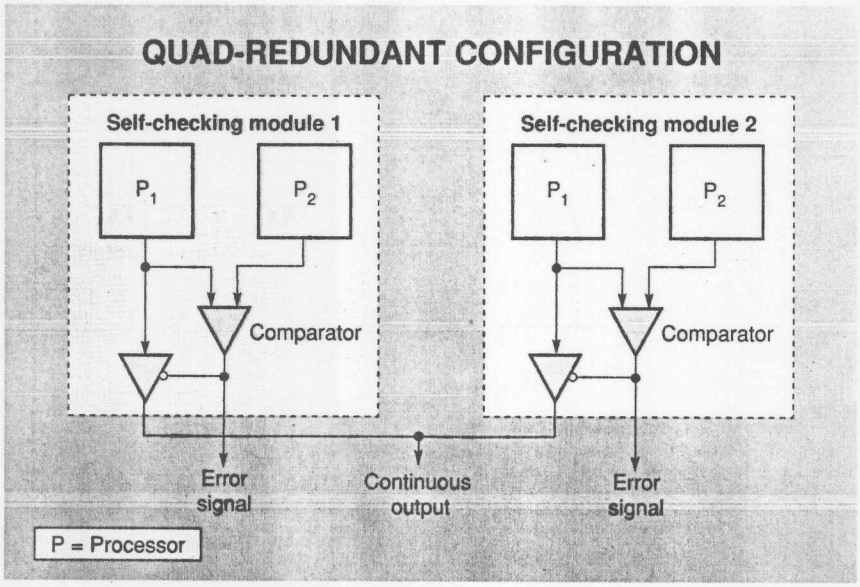

Figure 3: *This configuration is used for continuous error-free operation. A detected error disables the output of a faulty module, letting the other module continue.*

Simple parity checking uses a single redundant bit to provide a minimum separation of two. Words with even parity have an even number of 1 bits; therefore, a single error produces a word with an odd number of 1 bits, which identifies it as a noncode word.

Hamming codes (often used to protect memory systems) compute multiple parity bits for overlapping subsets of the bits within each data word. For a single error-correcting code, the overlap provides a minimum separation of three, enabling error correction.

Figure 4 illustrates a memory system utilizing an error-detection and -correction circuit. Check bits are computed and stored with the data during each memory write, and they are rechecked during each memory read; the data is corrected if errors are found.

Cyclic-redundancy-check codes commonly protect devices and communications channels that use serial data transfers. In CRCs, linear-feedback shift

registers compute a set of check bits over an entire string of data and then store or transmit the check bits after the data. The same operation is performed when retrieving or receiving the information; the computed check bits are compared to the original check bits to detect errors. Some CRC codes, such as those used on some high-performance disk drives, also provide sufficient redundancy to correct a limited number of bit errors.

Other Error-Detection Mechanisms

The state of a digital system in the clock period following the current one is a function of only its current state and the system inputs. In any particular state, the number of "next" states and inputs that can occur is relatively small. Hence, special hardware or software can often detect an improper input or an incorrect next state.

Several computer networks and system buses, especially those used in military and aerospace systems, are designed to ensure proper protocols during data transfers. They detect out-of-sequence or late-arriving events as system errors.

Typically, the limits on how long it can take to perform a particular event are known. Special watchdog timers can determine when an event fails to occur within its time frame and signal that problems exist. A wide variety of systems include time-out checks as an inexpensive way of detecting system failures, since such failures typically prevent an event from completing within its given time limit.

On several computers, operating-system software implements other protocol checks to ensure that the application programs follow proper procedures. In addition, most computers use special hardware to detect such errors as divide by zero, improper memory access, and nonexistent op codes. Most of these devices are relatively inexpensive to implement, and they often supplement other error-detection mechanisms in a fault-tolerant system.

The Three Rs

For continuous system operation, a fault-tolerance strategy must include the three Rs: repair, reconfiguration, and recovery. You must render a faulty component unable to affect other system elements

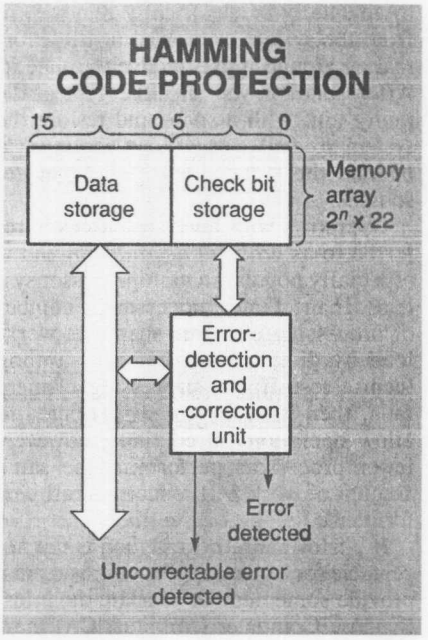

Figure 4: *A 16-bit memory array protected by a Hamming code. Six check bits enable single-error correction and double-error detection.*

by manually or electrically removing it from the system or by routing all information around it to effectively isolate it. After isolation, you can either replace the faulty unit with a spare and restore the system to full strength or continue to operate the system, but with fewer resources.

Operating with fewer resources is referred to as *graceful degradation* and is especially popular in multiprocessor systems. In most multiprocessors, a number of processing elements share the workload by distributing the tasks among themselves. If one processing element fails, then its tasks are redistributed to allow operations to continue; however, fewer processors performing the same amount of work will reduce overall performance.

If performance degradation is not acceptable for a given application, you must provide some number of spare modules. Synapse Computer (Milpitas, CA) marketed a multiprocessor called the N+1 system; it provided $n+1$ processing elements for applications requiring n processors to achieve the desired performance.

The N+1 system stored all tasks in a queue in a common memory area. Each processor continuously selected a new task from the queue after completing its current task. If any single processor failed, it was disabled and its task reentered on the queue. The remaining n processors continued to select tasks from the queue, ensuring continued correct operation.

Once a system has been restored to full strength or reconfigured to isolate a faulty unit, its operational state must be set to a correct value. The extent of the recovery depends on the extent of the error propagation. The most common approach is to restore, or roll back, the state of the system to a known good value.

Current Trends

With the ability to put more devices onto a single chip, VLSI designers are incorporating on-chip fault-detection mechanisms to improve testability for initial part checkout, to support diagnostic operations, and to provide on-line error detection during normal operations.

Additional on-chip features to support fault-tolerant system design are also beginning to appear, such as comparators at output pins to support duplex operations. Many current memory chips include on-chip logic to reconfigure the rows and columns of a memory array to isolate faulty storage cells (see "Chips That Work" on page 187). This is normally done at initial testing time to increase yield by eliminating manufacturing defects. In some cases, faults occurring during normal operation can be tolerated in the same manner.

While fault-tolerant design principles were once limited to special-purpose systems that had to be highly dependable, their use is beginning to extend to general-purpose minicomputers and mainframes, as can be seen in current offerings from IBM and DEC. This trend will also continue into the personal computer arena, making fault tolerance an integral and cost-effective part of all computer systems. ■

Victor P. Nelson is an associate professor in the electrical engineering department at Auburn University (Auburn, AL). He has a Ph.D. from Ohio State University. He can be reached on BIX c/o "editors."